Logic synthesis from concurrent specifications

Χρήστος Π. Σωτηρίου
Ινστιτούτο Πληροφορικής – ITE
Τμήμα Επιστήμης Υπολογιστών-
Πανεπιστήμιο Κρήτης

In collaboration with J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev

Outline

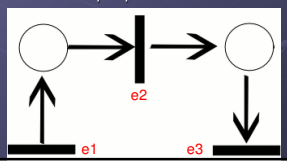
- Background
- Overview of the synthesis flow
- Specification
- State graph and next-state functions
- State encoding
- Implementability conditions
- Speed-independent circuit
  - Complex CMOS gates
  - C-element architecture
- Review of some advanced topics

Books and synthesis tool

- J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Logic synthesis for asynchronous controllers and interfaces, Springer-Verlag, 2002
- Jens Sparso and S. Furber, Principles of Asynchronous Circuit Design: A Systems Perspective, Kluwer Academic, 2002
- Petrify tool: http://www.lsi.upc.es/petrify

Background

- Petri-Net
  - graph-based mathematical model
  - place and transition nodes
  - directed arcs connecting places to transitions and transitions to places
  - places hold tokens
  - transitions "fire" tokens and move them from place to place
  - transitions may be enabled or not (e1, e2, e3)
- Petri-Net Properties
  - **Liveness** : every transition will fire infinitely
  - **Boundedness**: tokens per place



Background

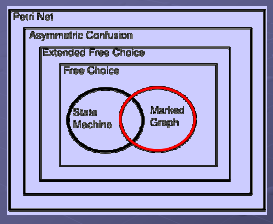
- Petri-net model can be reduced to:
  - **State Machine**
    - **exactly 1 incoming and 1 outgoing arc per transition**
    - **NO CONCURRENCY (never 2 tokens active)**
    - **CONFLICT!**
  - **Marked Graph**
    - **exactly 1 incoming and 1 outgoing arc per place**
    - **NO CONFLICT (never 2 choices possible)**
    - **CONCURRENCY!**

State Machine constrained Petri-net

Marked-Graph constrained Petri-net

Background

- Petri-net Taxonomy

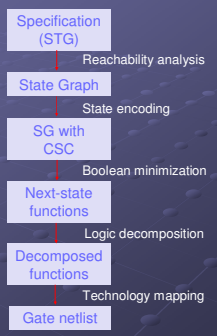


Marked Graph

Shorthand

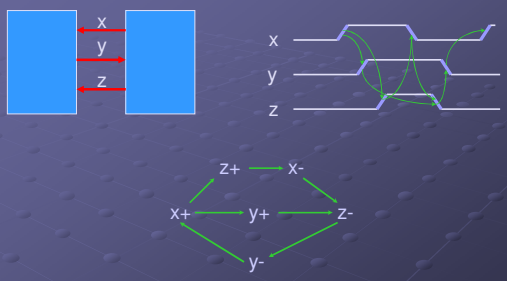
- Signal Transition Graph (STG)
  - Free-choice Marked Graph
  - Places hidden for Visibility, only transitions shown
  - Transitions correspond to signal changes, x+ or x-

Design flow



7

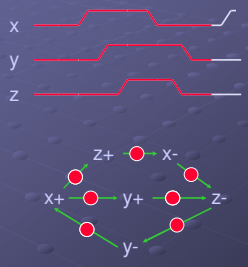
Specification



Signal Transition Graph (STG)

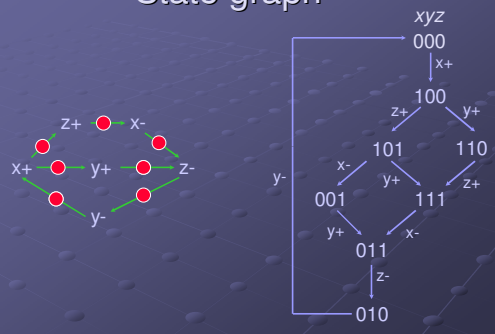
8

Token flow



9

State graph



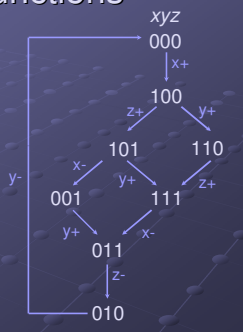
10

Next-state functions

$$x = \overline{z} \cdot (x + \overline{y})$$

$$y = z + x$$

$$z = x + \overline{y} \cdot z$$



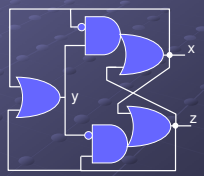
11

Gate netlist

$$x = \overline{z} \cdot (x + \overline{y})$$

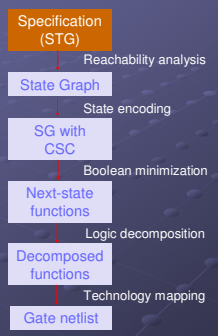
$$y = z + x$$

$$z = x + \overline{y} \cdot z$$



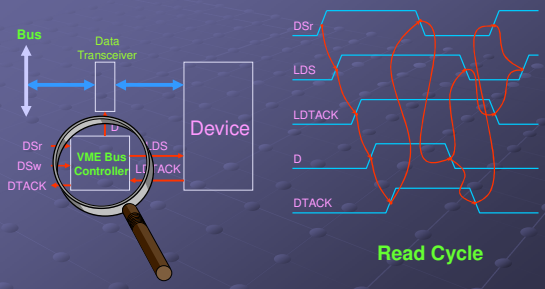
12

Design flow



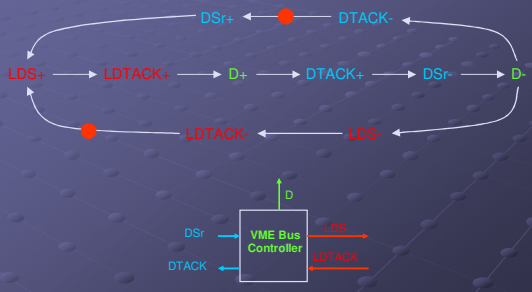
13

VME bus



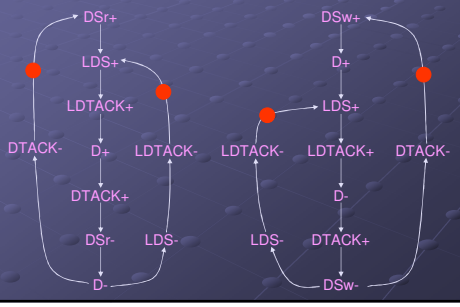
14

STG for the READ cycle



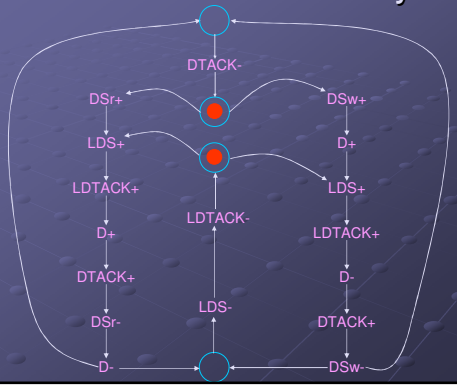
15

Choice: Read and Write cycles



16

Choice: Read and Write cycles



17

Circuit synthesis

- Goal:
  - Derive a hazard-free circuit under a given delay model and mode of operation

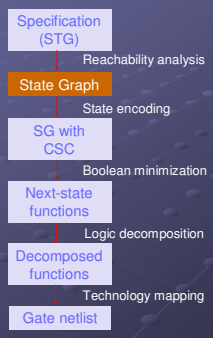
18

Speed independence

- Delay model
  - Unbounded gate / environment delays
  - Certain wire delays shorter than certain paths in the circuit
- Conditions for implementability:
  - Consistency
  - Complete State Coding
  - Persistency

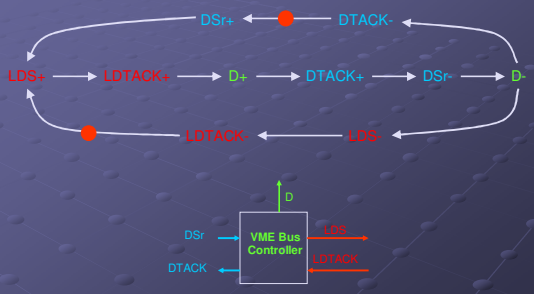
19

Design flow



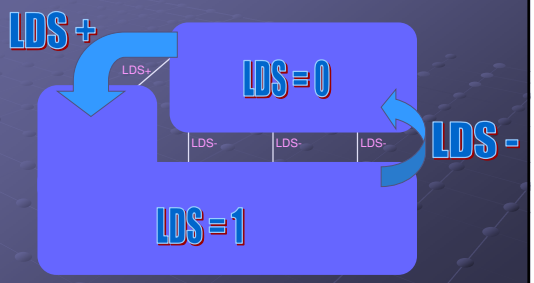
20

STG for the READ cycle



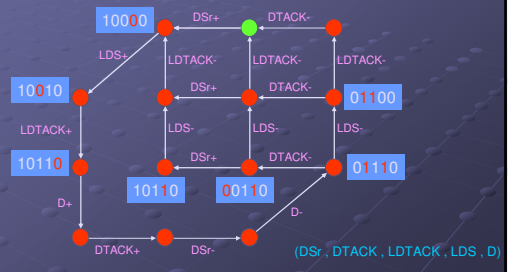
21

Binary encoding of signals



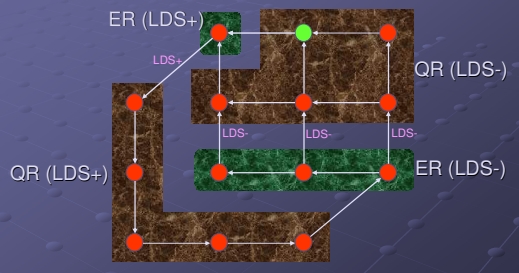
22

Binary encoding of signals



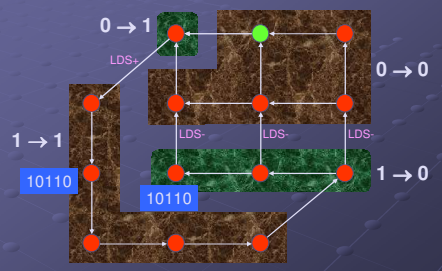
23

Excitation / Quiescent Regions



24

Next-state function



25

Karnaugh map for LDS

LDS = 0

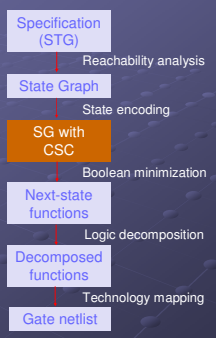
D LDTACK \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	0

LDS = 1

D LDTACK \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	0/1?

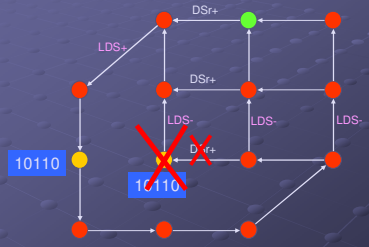
26

Design flow



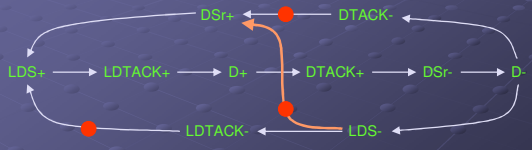
27

Concurrency reduction



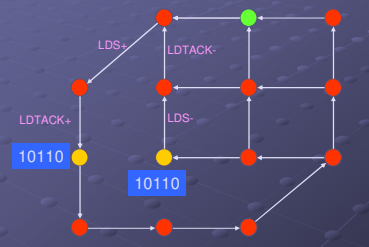
28

Concurrency reduction



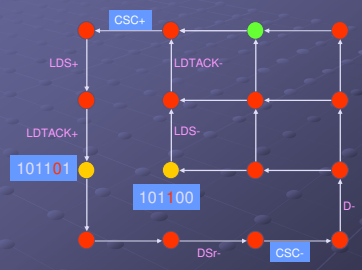
29

State encoding conflicts



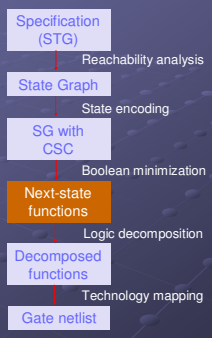
30

Signal Insertion



31

Design flow



32

Complex-gate implementation

$$LDS = D + \text{csc}$$

$$DTACK = D$$

$$D = LDTACK \cdot \text{csc}$$

$$\text{csc} = DSr \cdot (\text{csc} + \overline{LDTACK})$$

33

Implementability conditions

- Consistency
  - Rising and falling transitions of each signal alternate in any trace
- Complete state coding (CSC)
  - Next-state functions correctly defined
- Persistency
  - No event can be disabled by another event (unless they are both inputs)

34

Implementability conditions

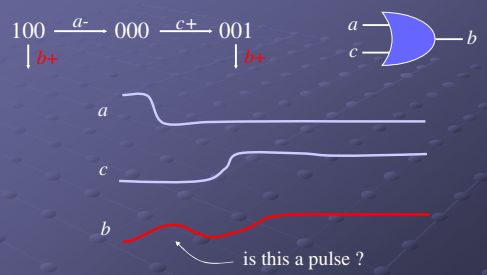
- Consistency + CSC + persistency

- There exists a speed-independent circuit that implements the behavior of the STG

(under the assumption that ay Boolean function can be implemented with one complex gate)

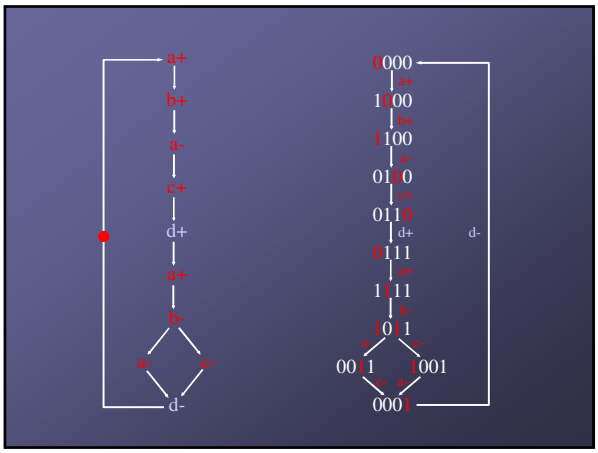
35

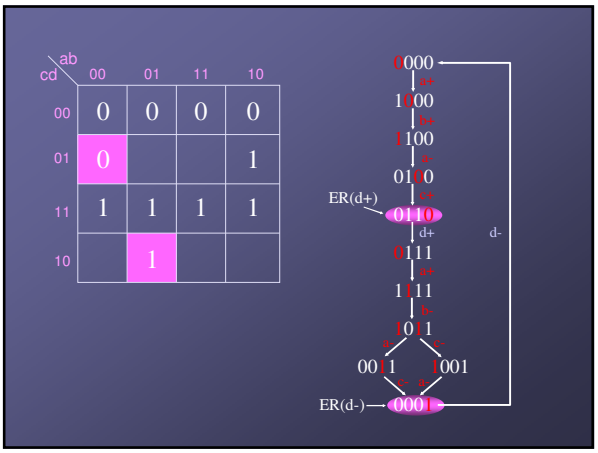
Persistency

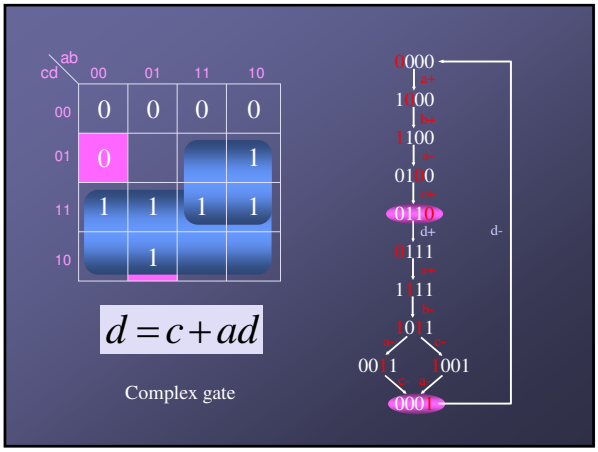


Speed independence ⇒ glitch-free output behavior under any delay

36





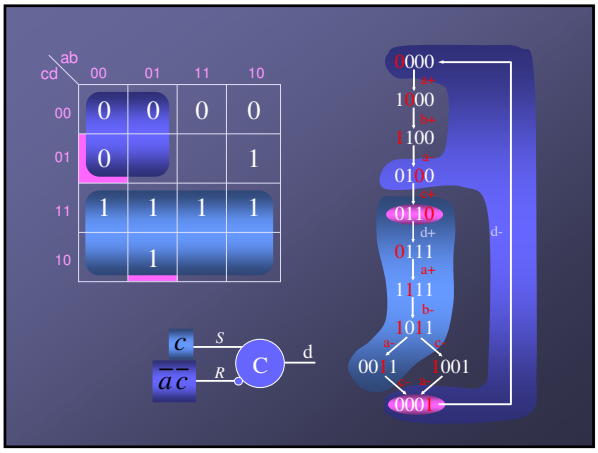


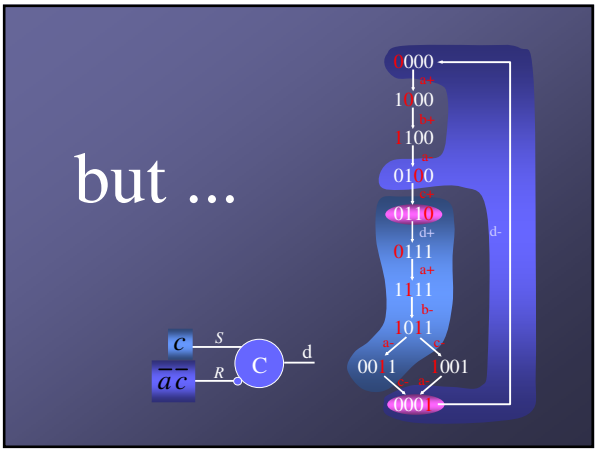
Implementation with C elements

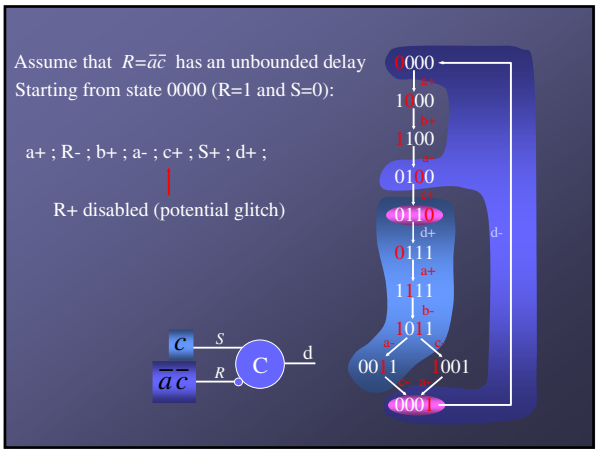
$\bullet\bullet\bullet \rightarrow S+ \rightarrow z+ \rightarrow S- \rightarrow R+ \rightarrow z- \rightarrow R- \rightarrow \bullet\bullet\bullet$

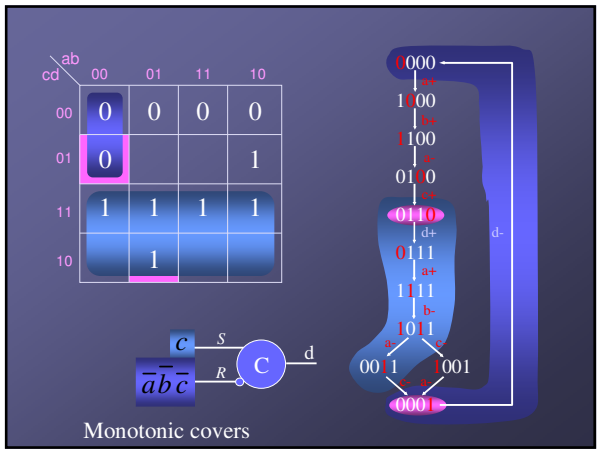
- *S* (set) and *R* (reset) must be mutually exclusive
- *S* must cover $ER(z+)$ and must not intersect $ER(z-) \cup QR(z-)$
- *R* must cover $ER(z-)$ and must not intersect $ER(z+) \cup QR(z+)$

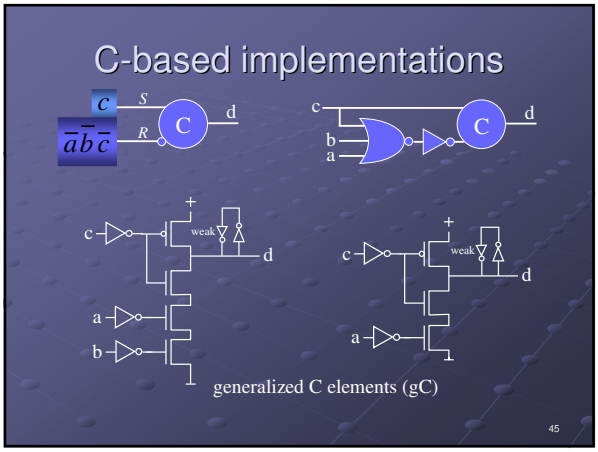
40

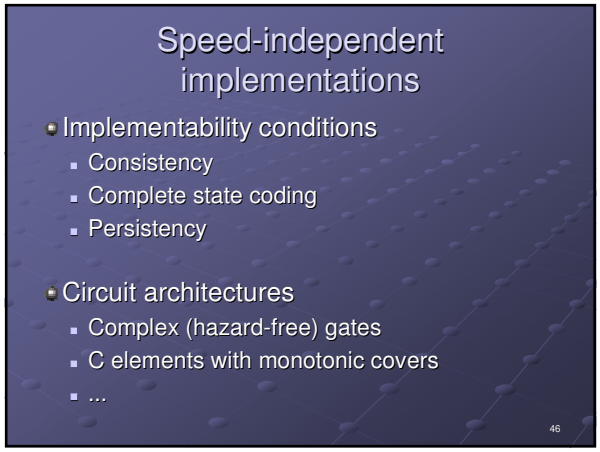


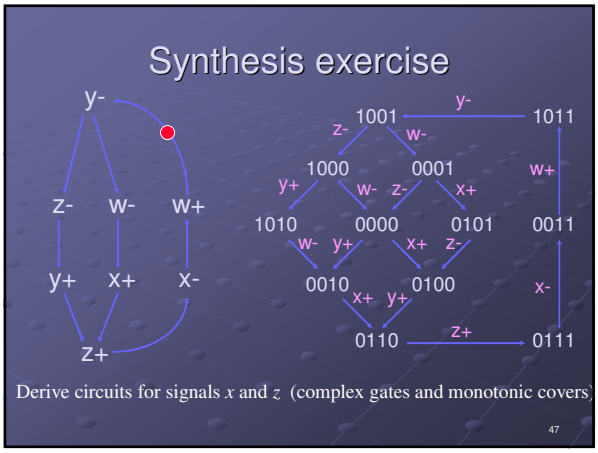


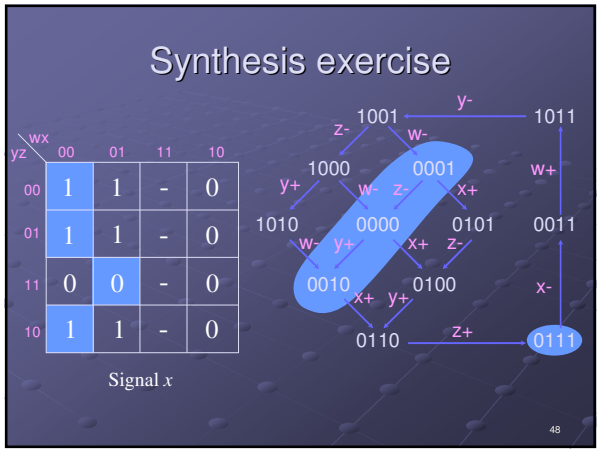

Assume that $R=\bar{a}\bar{c}$ has an unbounded delay
Starting from state 0000 (R=1 and S=0):
a+ ; R- ; b+ ; a- ; c+ ; S+ ; d+ ;
R+ disabled (potential glitch)
0000
1000
1100
0100
0110
0111
1111
1011
0011
1001
0001
c
S
R
C
d
$\bar{a}\bar{c}$


ab
cd
00 01 11 10
0 0 0 0
0 1
1 1 1 1
1
0000
1000
1100
0100
0110
0111
1111
1011
0011
1001
0001
c
S
R
C
d
$\bar{a}\bar{b}\bar{c}$
Monotonic covers

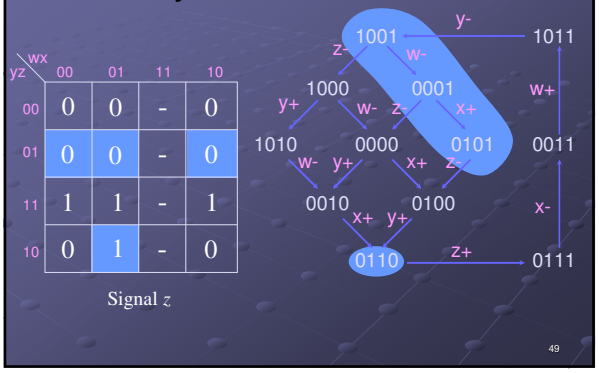

C-based implementations
c
S
R
C
d
$\bar{a}\bar{b}\bar{c}$
b
a
weak
generalized C elements (gC)
45


Speed-independent implementations
Implementability conditions
Consistency
Complete state coding
Persistency
Circuit architectures
Complex (hazard-free) gates
C elements with monotonic covers
...
46


Synthesis exercise
y-
z-
w-
w+
y+
x+
x-
z+
1001
1011
1000
0001
1010
0000
0101
0010
0100
0110
0111
Derive circuits for signals x and z (complex gates and monotonic covers)
47

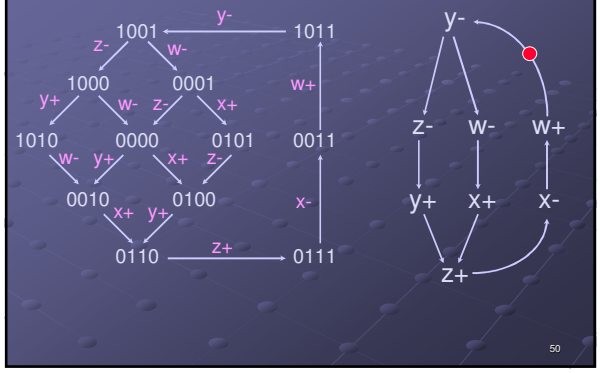

Synthesis exercise
wx
yz
00 01 11 10
1 1 - 0
1 1 - 0
0 0 - 0
1 1 - 0
Signal x
1001
1011
1000
0001
1010
0000
0101
0010
0100
0110
0111
48

Synthesis exercise



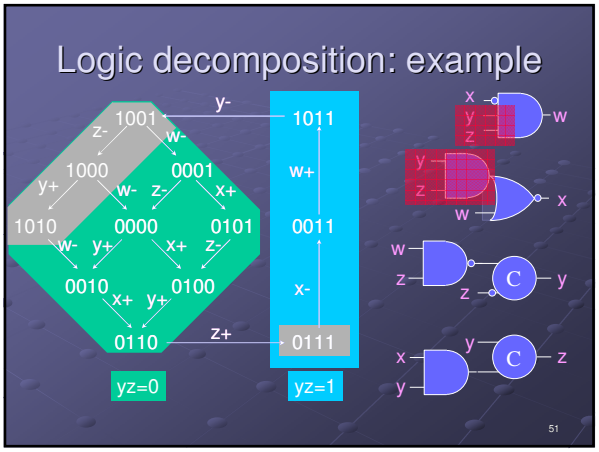
49

Logic decomposition: example



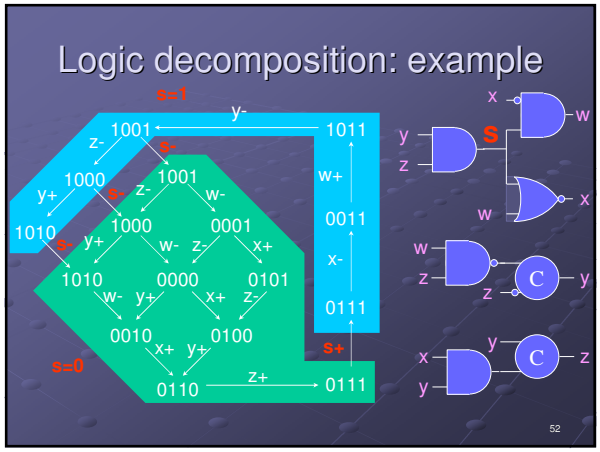
50

Logic decomposition: example



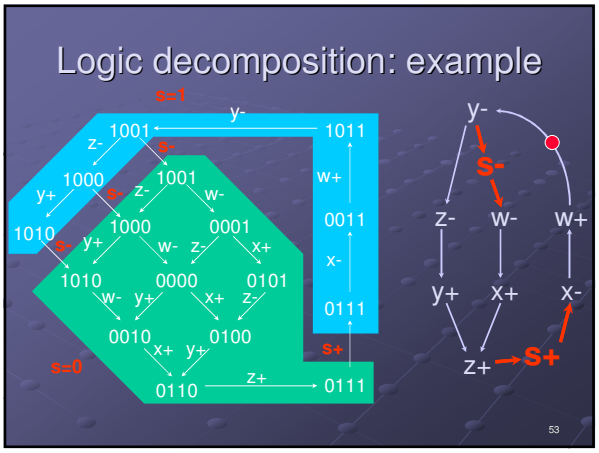
51

Logic decomposition: example



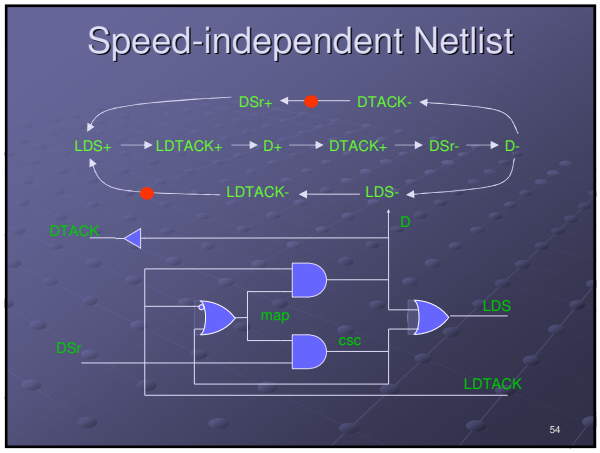
52

Logic decomposition: example



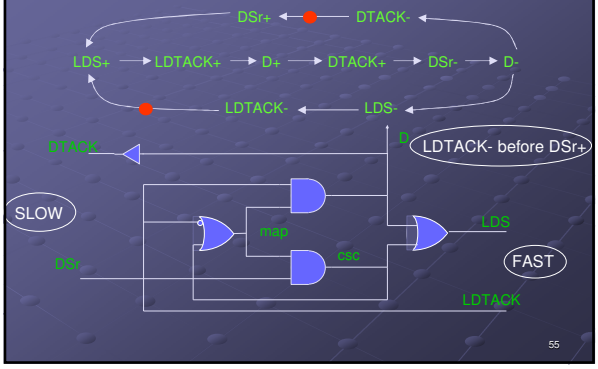
53

Speed-independent Netlist



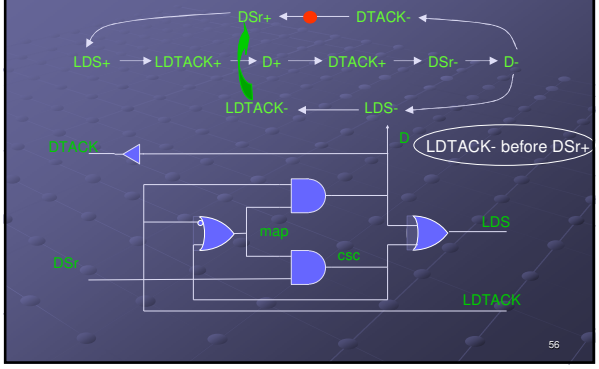
54

Adding timing assumptions



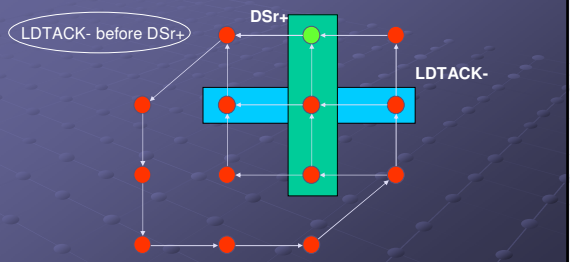
55

Adding timing assumptions



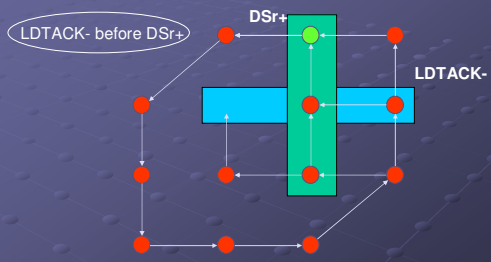
56

State space domain



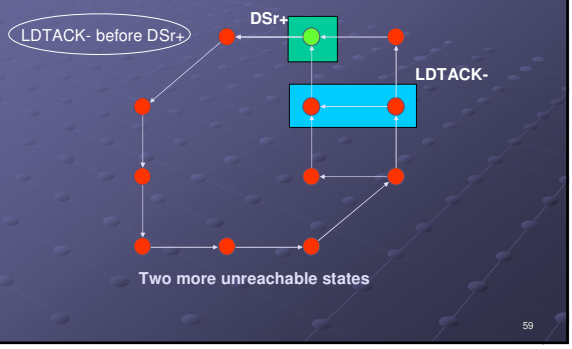
57

State space domain



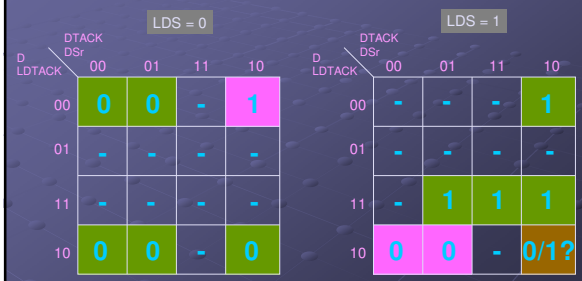
58

State space domain



59

Boolean domain



60

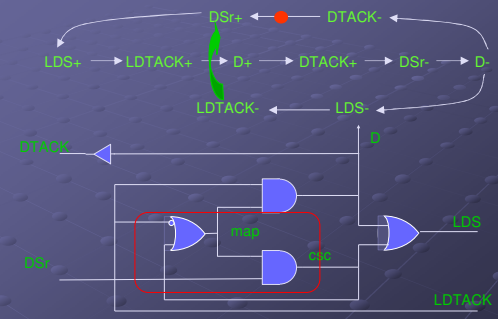
Boolean domain

One more DC vector for *all* signals

One state conflict is removed

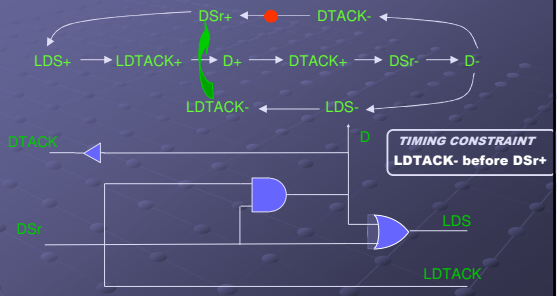
61

Netlist with one constraint



62

Netlist with one constraint



63

Conclusions

- STGs have a high expressiveness power at a low level of granularity (similar to FSMs for synchronous systems)
- Very effective approach for asynchronous control circuit design
- Not really suitable for datapath design, e.g. an adder
- Circuits with choice require attention for determinism (no confusion!)
- Synthesis from STGs can be fully automated
- Synthesis tools often suffer from the state explosion problem (symbolic techniques are used)

64